Algorithm Clustering for Multi-algorithm Processor Design

Madhushika M. E. Karunarathna [1], Yu-Chu Tian [2], Colin Fidge [3], Ross Hayward [4]
Queensland University of Technology, Brisbane 4000, Australia
[1] m.amarasinghearachchilage@student.qut.edu.au, [2] y.tian@qut.edu.au
[3] c.fidge@qut.edu.au, [4] r.hayward@qut.edu.au

***Abstract*—An Application Specific Instruction-set Processor (ASIP) is a specialized processor tailored to run a particular application/s efficiently. However, when there are multiple candidate applications in the application's domain it is difficult and time consuming to find optimum set of applications to be implemented. Existing ASIP design approaches perform this selection manually based on a designer's knowledge. We help in cutting down the number of candidate applications by devising a classification method to cluster similar applications based on the special-purpose operations they share. This provides a significant reduction in the comparison overhead while resulting in customized ASIP instruction sets which can benefit a whole family of related applications. Our method gives users the ability to quantify the degree of similarity between the sets of shared operations to control the size of clusters. A case study involving twelve algorithms confirms that our approach can successfully cluster similar algorithms together based on the similarity of their component operations.**

I. Introduction

Application-Specific Instruction-set Processors (ASIPs) are designed around operations executed frequently by a particular algorithm. Dedication of the processor core to a specific target application domain is provided by a customized Instruction Set Architecture (ISA). Customized instructions are tailored to execute "special-purpose operations" characteristic of the target domain.

To obtain expected performance it is important to select the best set of custom instructions for the given application/s. However the cost (time and complexity) involved in this task has become a bottle neck for the fast implementation of ASIP designs. If we are to design an ASIP system for an application domain it is necessary to compare the performance of each custom operation of all the objective applications with each other in order to select the best set under area and power constraints. In addition, it is necessary to maximize the reusability of the custom instructions so that a good trade-off between flexibility and high performance is maintained. Reusability can be analyzed by considering common special-purpose operations and hence custom instruction usage between applications. When there are multiple applications to be considered, various application combinations and their ISAs need to be generated and compared.

In many situations, however, it is not feasible to consider all operations and possible ISAs or to compare their performance with each other since the number of comparisons increases rapidly with the number of candidate applications. In existing multi-application based ASIP design methodologies, when many candidate applications are present, the target applications are selected manually based on prior knowledge of their properties [1]. However with a large number of candidate applications this process becomes unwieldy.

In response, we present an automated clustering method based on the applications' characteristic special-purpose operations, as an aid in reducing the number of candidate applications. In our method, the given applications are clustered based on common operations they share. Each cluster consists of a "representative application" which represents the custom instructions the particular cluster uses. Consequently, comparisons can be carried out between these representative applications instead of the whole group, which allows a significant reduction of the comparison overhead.

To do this, first the given set of applications are analyzed to identify their special-purpose operations, i.e., those sequences of primitive actions which are peculiar to, or best define, the application. The applications are then listed against all such operations found within the whole application collection. In this way each candidate application is represented as a point in a multidimensional space. The coordinates of each point are determined based on the presence of the special-purpose operations in the relevant application. By calculating the distance between each point using Euclidean geometry the differences between each pair of applications are then quantified. Based on these results application clustering is performed using a new classification approach, developed from the concepts of K-means clustering, designed to maximize the number of applications in each of the clusters in order to reduce the number of candidate applications efficiently.

II. Related Work

According to the literature many researches have been pay attention to identify common special-purpose operations when designing custom instructions for multi-application based specific processors [1], [2], [3]. However the main drawbacks of these approaches are selecting common operations based on previous knowledge, using manual methodology and lack of completeness as a method.

More generally, algorithm classification has been used to help understand the properties of algorithms and predict their behaviour [4], so many algorithm classification methods have been introduced in various fields [5], [6], [7], [8]. Although

all the above approaches produced good results they are not suitable to use in our purposes because of their limitation for specific application domains and parameters.

By contrast, our approach does not consider algorithm properties such as predicted execution times, data usage, etc, nor does it rely on subjectively-assessed properties. Instead it clusters algorithms purely on the basis of shared functionality, expressed in terms of the 'special-purpose operations' that best characterize the algorithms. This is done not to help understand the algorithms or predict their behavior, but to determine which algorithms could benefit from having certain operations available as primitives in the run-time processor.

III. Methodology

As shown in Figure 1, our application clustering process consists of six steps. It takes a candidate set of applications in a high level language (or equivalent pseudo code) as input and produces groups of applications clustered by the characteristic special-purpose operations they have in common. A detail description of each step in our methodology is presented in the following sub sections.

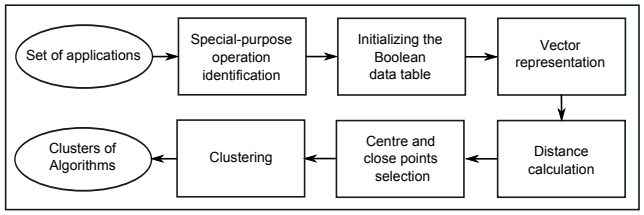

Fig. 1. Overview of the application clustering method

A. Special-Purpose Operation Identification

The first step is to identify the special-purpose operations that best characterize each application's purpose and in which it is expected to devote of its most execution time. In practice these operations are typically nested within loops, since this is where most computation time is spent, as per traditional application complexity analysis. When designing a special-purpose processor, these 'basic operations' are those that will prove most profitable to implement as primitive actions in hardware. To identify special-purpose operations either static or dynamic analysis methods can be used. For our case study (described in Section IV) static analysis strategies were used, following the established principles of complexity analysis [9]. Here algorithms are implemented in C/C++ and profiled statically to identify their characteristic operations. We then use the presence of shared operations to measure the similarity between different algorithms.

B. Initializing the Boolean Data Table

Once the special-purpose operations of each algorithm are identified (Section III-A) the presence of these operations is listed against each individual algorithm. Since we have already identified these operations as significant to run-time performance, we merely need to note their presence in each algorithm, so a Boolean table is sufficient.

C. Vector Representation

From the Boolean representation of each applications' characteristics (Section III-B) we produce a vector representation that can be used for distance calculations. If each operation in the list is considered as an independent variable, each application can be represented as a multi-variable equation P such that

$$P = Aa + Bb + Cc + \cdots = (A, B, C, \dots)$$

where A, B, and C are coefficients, which are either 1 or 0.

D. Distance Calculation

These vectors can be represented as points marked in a multi dimensional space according to their coordinates. In a multi dimensional space, the distance between two points is found by calculating the Euclidean distance between them. Thus, the difference between two algorithms can be represented by a numerical value calculated using the following equation.

Definition 1 (Euclidean distance): If points A and B can be represented as an equation of n independent variables then the Euclidean distance of A and B, denoted by D_{AB}, is:

$$D_{AB} = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_n - b_n)^2}\,.$$

E. Centre and Close Points Selection

Whereas some clustering techniques begin from a random point, we instead identify *centre points* first, which are surrounded by many other points called the *close points*. This reduces the complexity of the classification by avoiding repeated iterations to find centres and members and comparisons between formed groups. Since we aim to maximize the number of algorithms in each cluster, the centre points which fulfil the requirements of a successful centre form "clusters" along with their close points. To determine close points, a *distance constant* value, DC, is used as the maximum distance between a centre point and its close points.

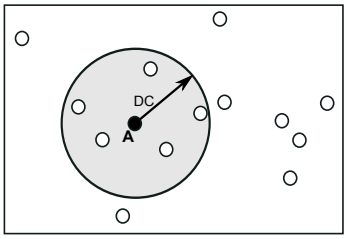

Fig. 2. Close points to A as per distance constant DC

Since the difference between each application pair is measured by the distance of their sets of special-purpose operations, the DC value is calculated based on the expected number of different operations between the pairs (or conversely the number of shared operations). Thus it is a threshold value to represent the maximum expected difference between a cluster's centre application and its member applications. The maximum distance possible between two points, and hence two applications, in the same cluster will be $2 \times DC$.

Initially all the points in the space are considered as potential centre points. Then for each centre point its close points which are located less than DC units away from the centre point are found. These points are added to each centre point's 'close point list' in order to form clusters in the next step. In our approach the user is allowed to choose a suitable DC value based on the desired coherence of the cluster. For instance, if a user expects to allow $p\%$ minimum similarity of special-purpose operations between a cluster's centre and its member points, the DC threshold can be calculated as follows.

Definition 2 (Number of shared operations): When the total number of operations considered is denoted by O_n, then the required number of operations to be shared, O_s, is

$$O_s = p \times O_n/100.$$

Definition 3 (Distance constant): The distance constant is

$$DC = \sqrt{O_n - O_s}.$$

The DC threshold can vary depending on the number of applications, the number of special-purpose operations in the set of applications and their expected behaviour. Thus, users can decide on a suitable similarity percentage and hence calculate DC for the particular application set of interest.

F. Clustering

Clustering begins from the centre point which has the longest close point list. However, sometimes more than one centre point can have the same highest number of points in its group. In that case, a *priority list* is generated in order to determine the first centre point using the following criteria. First, the total of all Euclidean distances between all pairs of points in the groups are calculated. This can be used to measure the similarity of the group. If the total distance of a group is small, the difference between any two applications in the group is likely to be small. Therefore, this group is more cohesive and has a higher priority for becoming a useful cluster. Thus, the groups which have low values for their total distance will have higher cohesiveness and higher priority. Then, the centre point which owns the highest priority group is used as the first starting point.

The first centre point along with its close points forms a cluster. Then all members of this newly created cluster are removed from the centre point list. They cannot form the centre point of another cluster since they are already in a cluster. In addition, all the members are removed from other points' close point lists in order to avoid overlapping clusters.

That is, a point can be a member of only one cluster. Then the centre point which has the longest close point list will be the next centre point of the newly modified centre point list after these eliminations, and so on. If there is more than one candidate centre point, the same criteria as above are followed to determine the priorities of each point and hence find the next centre point. Clustering thus continues until all the points in the centre point list are covered or removed.

The clusters obtained in this step represent the final set of application groupings. The corresponding application clusters can form the target application domain for a multiple application-enabled special-purpose processor, based on the commonality of their operations.

IV. Practical Results

In order to validate our application classification methodology we have performed a case study on the *exact string matching* application domain. A tool was developed to support the clustering process automatically in C/C++, once the vectors representing the applications have been defined. The analysis was done on a set of twelve string matching algorithms.

String matching algorithms are used to identify all occurrences of a given substring, known as a *pattern*, in a given text domain [10]. Many crucial areas such as text processing, information retrieval, natural language processing and bioinformatics are based on string matching techniques.

For our case study, we chose the set of twelve commonly used exact string matching algorithms: *Morris-Pratt (MP), Knuth-Morris-Pratt (KMP), Apostolico-Crochemore (AC), Boyer-Moore (BM), Turbo Boyer-Moore (TBM), Zhu-Takaoka (ZT), Horspool (HP), Raita (RT), Colussi (CL), Galil-Giancarlo (GG), Not So Naive (NSN) and Apostolico-Giancarlo (AG).*

All these algorithms perform the same matching functionality in different ways. Therefore, since it is likely that many similar special-purpose operations are shared between algorithms, clustering them into different clusters is more difficult than clustering different native applications. However, our methodology can be easily applied when different kinds of applications are to be considered.

Table I lists few of twenty-six special-purpose operations found in the algorithms, indicating their absence or presence in the algorithms' (pseudo-)code by '0' or '1'.

Then as described in Section III-C and III-D the Euclidean distance matrix is obatined for the whole data set. The next task is selecting centre points in order to perform clustering. As described in Section III-E, first a DC value was calculated as the maximum allowed distance between centre points to their close points. In this case study we have specified the expected similarity between algorithms to be 90% and the total number of special-purpose operations is 26. Thus the DC value was calculated as 1.73 followith as per Definitions 2 and 3.

Then, using this DC value, close point selection was performed over the Euclidean distance matrix data. That is, for the first centre point in the list, which is in our case the MP algorithm, the other points which have a Euclidean distance less than 1.73 from MP's vector were considered. Likewise for all the points in the centre point list, their close points being identified as follows:

Centre	Close Points	Centre	Close Points
MP	KMP	HP	RT
KMP	MP, AC	RT	HP
AC	KMP	CL	GG
BM	TBM, ZT, AG	GG	CL
TBM	BM, AG	NSN	-
ZT	BM, AG	AG	BM, TBM, ZT

Algorithm clustering was then performed, considering the above centre points and their close points lists.

TABLE I
NUMBER OF OCCURRENCES OF SPECIAL-PURPOSE OPERATIONS IN THE CANDIDATE ALGORITHMS

Special-purpose Operations	MP	KMP	AC	BM	TBM	ZT	HP	RT	CL	GG	NSN	AG
1) $x[i] \neq x[j]$	1	1	1	0	0	0	0	0	0	0	1	0
2) $x[i]$ == $y[i+j]$	0	0	1	1	1	1	0	0	0	1	0	1
3) $x[i]$ == $x[i+m-1-n]$	0	0	0	1	1	1	0	0	0	0	0	1
4) i = $x[m]$ + 1	0	0	0	1	0	1	1	1	1	1	0	0
5) for (i = 0; $i < m$; ++i), $x[i]$ = n;	0	0	0	1	1	1	1	1	1	1	0	1

The first cluster of this case study was thus the Apostolico-Giancarlo algorithm and its close points. Clustering was carried out until all twelve algorithms were counted. Figure 3 shows the four major algorithm clusters ultimately obtained.

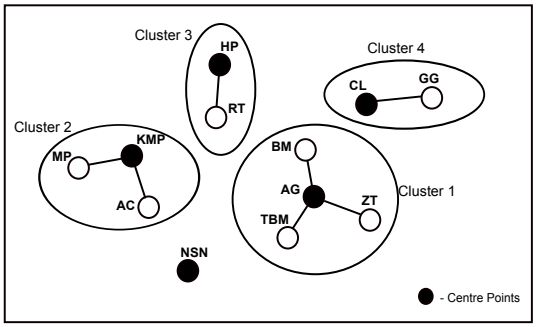

Fig. 3. Resultant algorithm clusters

V. CONCLUDING REMARKS

As a final validation of our clustering approach we can consider how the grouped algorithms are related historically. Cluster 1 in Figure 3 involves the BM, TBM, ZT and AG algorithms. As TBM is an extension of BM, these two algorithms share the same special-purpose operations. ZT is a variant of the BM algorithm and performs shifting by considering the 'bad-character shift' introduced in the BM approach for two consecutive text characters [11]. The AG algorithm is also a variant of BM; it addresses the problem of forgetting characters it has already matched after each attempt [11]. Thus all of these algorithms clustered automatically by our methodology are truly related and use similar special-purpose operations to achieve their exact string matching functionalities.

Cluster 2 consists of the MP, KMP and AC algorithms. MP and KMP use similar special-purpose operations since KMP extends the MP algorithm. Also AC uses the KMP 'shift table' for window shifting. Thus, it is inherited by KMP and hence uses the same common special-purpose operations [11].

Cluster 3 is formed on the basis that the HP and RT algorithms utilize similar operations. According to Table I all the special-purpose operations found in both algorithms are similar and hence they coincide with each other. Both algorithms use a 'bad-character shift' in their pre-processing phase. Also, they both use memory compare operations in the searching phase [11]. Thus, they quite reasonably belong to the same cluster created by our method.

Cluster 4 is comprised of the CL and GG algorithms. The GG algorithm is a refinement of the CL algorithm [11]. The only difference is that it intervenes in the searching phase. Therefore these two algorithms also belong together.

The sole outlier is the NSN algorithm which uses a unique approach for string matching, such that preprocessing can be done in constant time and space [11]. This algorithm remained isolated since it does not share a similar pattern of special-purpose operations to any other algorithm.

This analysis shows that our algorithm clustering approach correctly clusters algorithms according to the special-purpose operations they share. Any such cluster can now be chosen as the basis on which to design a special-purpose processor, and doing this is the subject of our ongoing research.

REFERENCES

[1] L. Fanucci, M. Cassiano, S. Saponara, D. Kammler, E. Witte, and O. Schliebusch, "ASIP design and synthesis for non linear filtering in image processing," in *Proceedings of Design, Automation and Test in Europe, 2006. DATE '06*, vol. 2, Mar 2006, pp. 1–6.

[2] M. Arnold and H. Corporaal, "Designing domain-specific processors," in *Proceedings of the Ninth International Symposium on Hardware/Software Codesign, 2001*, 2001, pp. 61–66.

[3] K. Shirai, T. Ikenaga, and H. Kitabatake, "Design system for special purpose processor executing algorithms described by higher level language," in *Proceedings of Third Annual IEEE ASIC Seminar and Exhibit, 1990*, Sep 1990, pp. P7/1.1 –P7/1.4.

[4] J. Woodward and J. Swan, "Why classifying search algorithms is essential," in *IEEE International Conference on Progress in Informatics and Computing (PIC)*, vol. 1, Dec 2010, pp. 285 –289.

[5] Y.-K. Kwok and I. Ahmad, "Benchmarking the task graph scheduling algorithms," in *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*, Mar 1998, pp. 531 –537.

[6] Y.-S. Ong, M.-H. Lim, N. Zhu, and K.-W. Wong, "Classification of adaptive memetic algorithms: a comparative study," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 36, no. 1, pp. 141 –152, Feb 2006.

[7] R. Riedl and L. Richter, "Classification of load distribution algorithms," in *Proceedings of the Fourth Euromicro Workshop on Parallel and Distributed Processing*, Jan 1996, pp. 404 –413.

[8] F. D. A. Zampirolli and R. D. A. Lotufo, "Classification of the distance transformation algorithms under the mathematical morphology approach," in *Proceedings XIII Brazilian Symposium on Computer Graphics and Image Processing*, 2000, pp. 292 –299.

[9] A. Levitin, *Introduction to the Design and Analysis of Algorithms*, 2nd ed. Addison-Wesley, 2007, ISBN 0-321-36413-9.

[10] M. J. Fischer and M. S. Paterson, "String-matching and other products," Massachusetts Institute of Technology, Cambridge, MA, USA, Tech. Rep., 1974.

[11] C. Charras and T. Lecroq, *Handbook of Exact String Matching Algorithms*. King's College Publications, 2004.